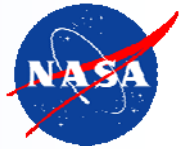
National Aeronautics and Space Administration



XTCE

XML Telemetry & Command Exchange Tutorial

XTCE Version 1.1 and GovSat

Kevin Rice, NASA GSFC, GST Inc., Kevin.Rice@gst.com

Brad Kizzort, Harris Corp., bkizzort@harris.com

Gerry Simon, Integral Systems, Inc., gsimon@integ.com

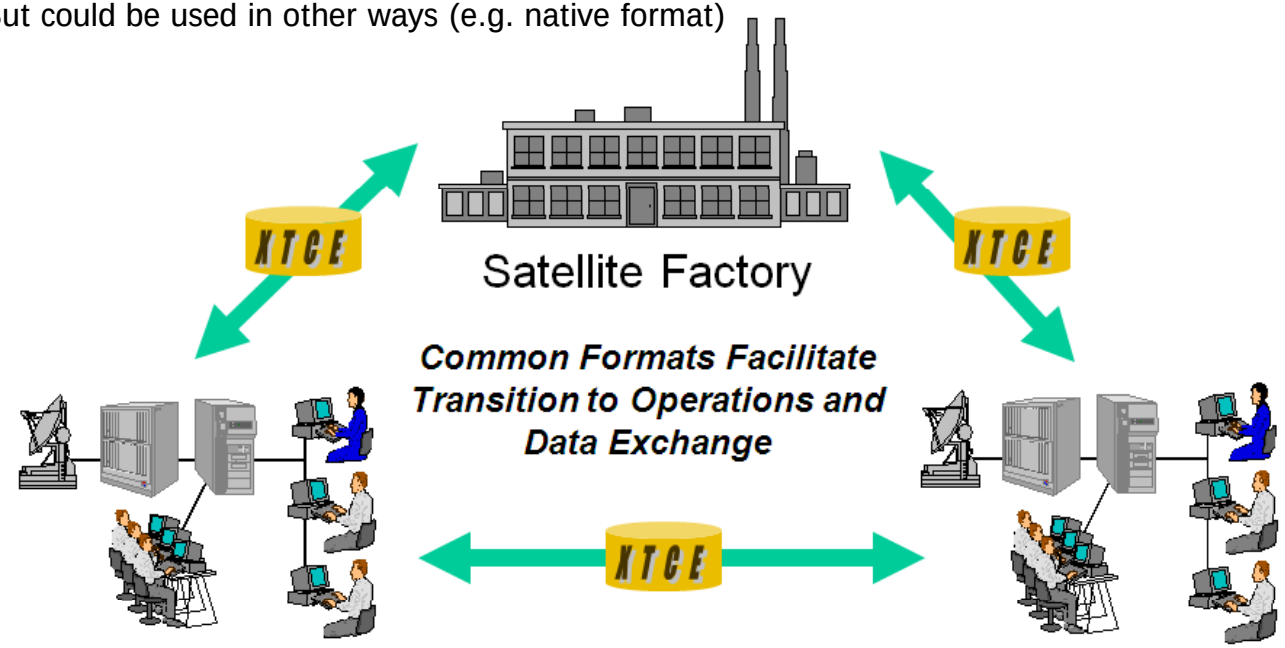
GSAW 2010

March 3, 2010

Contents

Preface - Goal

- Standard for describing telemetry and commanding
- Lower cost and increase validation (XML) over traditional formats
- Conceived as universal exchange mechanism between ground segment apps (users) that need TLM/CMD descriptions
  - But could be used in other ways (e.g. native format)



Preface - Brief History and Background

- XTCE 1.0 published in 2005 (OMG only)
- XTCE 1.1 published in Oct 2007 (CCSDS and OMG)
- XTCE 1.1 Green Book – overview, 2007
- XTCE 1.1 Magenta Books, under review
  - Core -- User Guide
  - CCSDS tailoring
- XTCE 1.2 – 1st draft review

Preface - Applicability

- XTCE is very flexible
  - Large # of elements and attributes to describe wide range of “blocky data”, but…
- This discussion will be principally oriented towards Packets or Minor Frames
- Assumes traditional tlm/cmd approach:
  - “bit-packed” data - no markers, metadata, non-self describing entries in data blocks
  - But some format markers (such as CCSDS packet header fields)
- Oriented towards Exchange
  - Ground applications that either need or produce telemetry and command descriptions
  - But are tied to native formats
  - Instead of many-to-many conversion
  - Each transfers to XTCE and back, resulting in…
    - More flexibility to mix and match ground apps
    - Economies of scale
    - Avoid proprietary formats
- Features in XTCE are needed by ground system apps
  - While every feature may not be needed, many features should be needed

Chapter 1 – The SpaceSystem

- The SpaceSystem is the root or the outermost element of any XTCE document
  - Can be used to represent almost any aspect of your architecture:
    - The entire project, a portion of your project, subsystems, etc…
  - Optional child SpaceSystem – build a tree like structure of SpaceSystems
  - Could be used in single spacecraft to entire constellation scenario

<xtce:SpaceSystem *attributes*>

<xtce:TelemetryMetaData/>

<xtce:CommandMetaData/>

<xtce:SpaceSystem attributes/>

</xtce:SpaceSystem>

Chapter 1 – The SpaceSystem

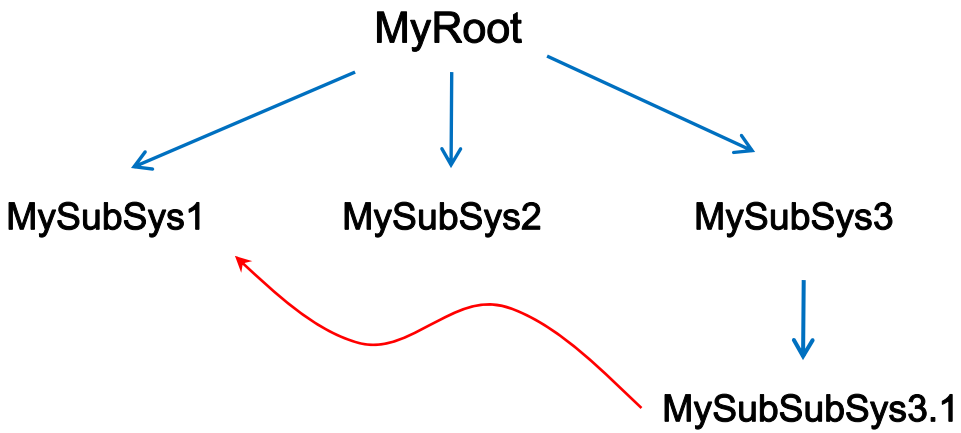

```
<xtce:SpaceSystem name="MyRoot">
  <xtce:TelemetryMetaData/>
  <xtce:CommandMetaData/>
  <xtce:SpaceSystem name="MySubSys1"/>
  <xtce:SpaceSystem name="MySubSys2"/>
  <xtce:SpaceSystem name="MySubSys3">
      <xtce:TelemetryMetaData/>
      <xtce:CommandMetaData/>
      <xtce:SpaceSystem name="MySubSubSys3.1"/>
   <xtce:SpaceSystem>
<xtce:SpaceSystem>
```

Common Tlm/Cmd

Tlm/Cmd for this sub-sys

Note: some required attributes or elements not shown for purposes of illustration

Chapter 1 – The SpaceSystem



Any **<SpaceSystem>** tree-structure is possible
-- Items defined in one **<SpaceSystem>** may refer to another using a string pointer ("Ref")
-- "Refs" come in various forms, similar syntax to directory paths
-- The simplest form is just the name of the item of interest, means local to current **<SpaceSystem>** or any above it
-- Absolute and relative path are also supported, go to the item the **<SpaceSystem>** specified
A single **<SpaceSystem>** with globally unique names of things is simplest

Chapter 1 – The SpaceSystem


```
<xtce:SpaceSystem>
    <xtce:TelemetryMetaData/>
    <xtce:CommandMetaData/>
    <xtce:SpaceSystem/>
</xtce:SpaceSystem>
```

- Each SpaceSystem has a telemetry and command description area
  - Both are optional
    - (a minimally valid XTCE file is a single SpaceSystem and name – more or less)
  - TelemetryMetaData – describe telemetry things here
  - CommandMetaData – describe command things here
  - A description in one may refer to a description in the other in which case
    - description is “co-opted” as if it were originally defined on that side
  - The name-space for TelemetryMetaData and CommandMetaData is shared
    - For example a Parameter defined in TelemetryMetaData is visible in CommandMetaData

Chapter 1 – Parameters and ParameterTypes


```
<xtce:SpaceSystem>
   <xtce:TelemetryMetaData>
      <xtce:ParameterSet/>
      <xtce:ParameterTypeSet>
   </xtce:TelemetryMetaData>
   <xtce:CommandMetaData/>
   <xtce:SpaceSystem/>
</xtceSpaceSystem>
```

- Telemetry parameters are defined with elements **<Parameter>** & **<ParameterTypes>**
  - (There are also command parameters, session or system parameters and derived or pseudo-parameters)
  - A parameter HAS A parameterType and some Properties
  - ParameterType descriptions may be shared with more than one Parameter
- Use these to describe the name of a single telemetry item and show its
  - Link Data Type → Conversion → Host Data Type relationship

Chapter 1 – Containers


```
<xtce:TelemetryMetaData>
   <xtce:ParameterSet/>
   <xtce:ParameterTypeSet/>
   <xtce:ContainerSet>
     <xtce:SequenceContainer/>
   </xtce:ContainerSet>
</xtce:TelemetryMetaData
```

- Use **<SequenceContainer>** to describe telemetry formats (CCSDS packet, etc…)
  - Specify Sequence of parameters
    - various options to manipulate sequence
  - Optionally **EXTEND** another container in an inheritance like fashion
    - Use element **<BaseContainer>** to name the parent container
    - Specify **<RestrictionCriteria>** to define identifying "keys" and expected values (such as APID=10)
    - Principally a construction mechanism, child inherits certain properties from the parent, mainly its entries
    - And also used to provide the identifying information
- (CommandContainers and MetaCommand/CommandContainers are similar)

Chapter 1 – Command and CommandContainers


```
<xtce:SpaceSystem>
   <xtce:TelemetryMetaData/>
   <xtce:CommandMetaData>
      <xtce:ParameterSet/>
      <xtce:ParameterTypeSet/>
      <xtce:ArgumentTypeSet/>
      <xtce:MetaCommandSet/>
      <xtce:CommandContainerSet/>
   </xtce:CommandMetaData>
   <xtce:SpaceSystem/>
</xtce:SpaceSystem>
```

Put Command related description in **<CommandMetaData>**

- Command **Parameter** and **ParameterType**
  - Same construction as on telemetry side but some child elements or should be ignored and are probably not applicable
- **ArgumentTypeSet** – command Arguments are similar to Parameters in constructions
- Command Description in element: **<MetaCommand>**
  - Included its arguments, local **<CommandContainer>** (packaging information, verifiers and other cmd related items
  - Command **<Argument>** links to **<ArgumentType>**

Chapter 1 – NameReferences

```
<SpaceSystem>
  <TelemetryMetaData>
    <ParameterTypeSet>
      <IntegerParameterType name="MyType"/>
    </ParameterTypeSet>
    <ParameterSet>
      <Parameter name="MyParam" parameterType="MyType"/>
    </ParameterSet>
  </TelemetryMetaData>
</SpaceSystem>
```

- NameReferences (NameRefs or Refs) are string pointers
  - Points from one location in an XTCE file to another, perhaps 50 NameRefs at various elements
  - Format: *{[SpaceSystem name, .., .]/}** itemName
    - myParameter
    - /SpaceSystemName1/SpaceSystemName2/myParameter
    - ../SpaceSystemName2/myParameter
  - The first (plain or unqualified) is <u>not</u> global; refers to local SpaceSystem first or up "the SpaceSystem tree" relative to that location
    - Favor relative addressing vs absolute if used
  - Outside of XML parsing unfortunately – you must implement it
    - Careful implementation required for general case
  - … (But only one SpaceSystem is easiest, use plain version for everything)


```
<xtce:SpaceSystem>
    <xtce:TelemetryMetaData>
<xtce:ParameterTypeSet/>
<xtce:ParameterSet>
</xtce:TelemetryMetaData>
    <xtce:CommandMetaData/>
    <xtce:SpaceSystem/>
</xtce:SpaceSystem>
```

Chapter 2 – ParameterTypes


```
<xtce:ParameterTypeSet>
  <xtce:StringParameterType/>
  <xtce:EnumeratedParameterType/>
  <xtce:BinaryParameterType/>
  <xtce:IntegerParameterType/>
  <xtce:FloatParameterType/>
  <xtce:BooleanParameterType/>
  <xtce:AbsoluteTimeParameterType/>
  <xtce:RelativeTimeParameterType/>
  <xtce:ArrayParameterType/>
  <xtce:AggregateParameterType/>
</xtce:ParameterTypeSet>
```

The names represent host side data types

String,
Enumerated,
Binary,
Integer
Float
Boolean
AbsoluteTime
RelativeTime
Array
Aggregrate (groups other Parameters)

These data types are what your system will process the data encoded on the link into

Advanced: a ParameterType may optionally inherit from another, see the @baseType attribute. Simpler implementations can ignore.

Chapter 2 – Pattern to (most) ParameterTypes


```
<xtce:StringParameterType>
    <xtce:StringDataEncoding/>
    <xtce:IntegerDataEncoding/>
    <xtce:FloatDataEncoding/>
    <xtce:BinaryDataEncoding/>
  <xtce:Alarms/>
</xtce:StringParameterType>
```

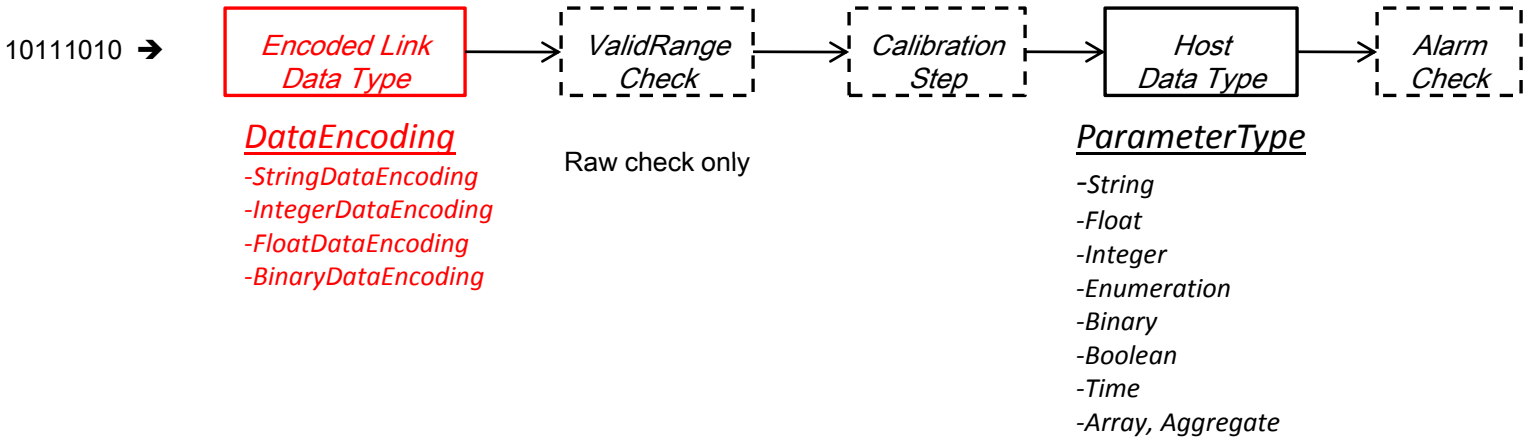
Choice of One or None

Pattern:

- DataType + (choice of one) DataEncoding
- DataEncodings describe transmitted (link) information
- Alarms and various details vary by ParameterType
- Likely common combinations of ParameterType and DataEncoding choices are documented
- Other combos may make sense (mission specific), and not strictly speaking illegal at this time

Chapter 2 – ParameterType Semantics

- ParameterTypes and their child elements and attributes, taken together, represent this relationship between information on the link and ground



Missions will likely wish to restrict these various elements and attributes for various ParameterType...

Chapter 2 - DataEncodings

- Four options, various details:
  - StringDataEncoding
    - UTF8 or UTF16 encoded Unicode (UTF-16 big endian)
    - Bit size, various ways
  - IntegerDataEncoding
    - Unsigned or TwosComplement (spelled “Compliment” in XTCE!)
    - Bit size
    - bit/byte order
    - Various calibrators such as Linear Calibrator or Polynomial Calibrator optional
  - FloatDataEncoding
    - IEEE-745, MIL1750A
    - Bit size
    - bit/byte order
    - Various calibrators such as Linear Calibrator or Polynomial Calibrator optional
  - BinaryDataEncoding
    - Bit size
    - bit/byte order
    - No cals...

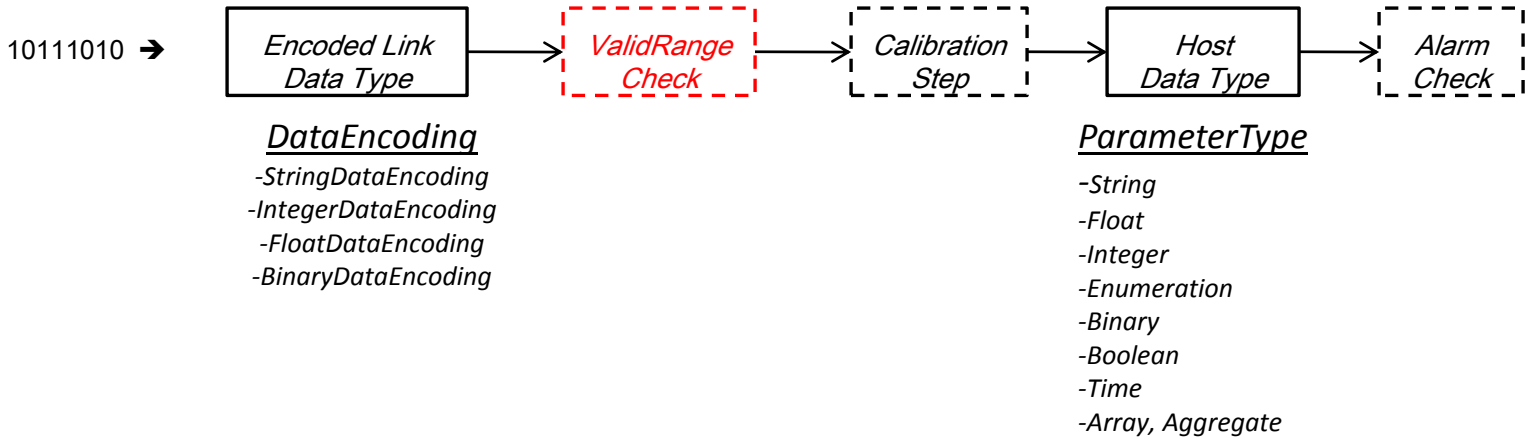
Some additional details left out

Chapter 2 - DataEncoding – Examples

Unsigned int, 32 bits	<xtce:IntegerDataEncoding sizeInBits="32"/>
Two Complement, 8 bits	<xtce:IntegerDataEncoding encoding="twosCompliment"/>
Float, 32-bit, IEEE-754	<xtce:FloatDataEncoding/>
Unicode String, UTF-16	<xtce:StringDataEncoding encoding="UTF-16"> <!-- Example: Length in bits is 11 characters --> <!-- 16-bits per character --> <xtce:SizeInBits> <xtce:Fixed> <xtce:FixedValue>176</xtce:FixedValue> </xtce:Fixed> </xtce:SizeInBits> </xtce:StringDataEncoding>

Note: defaults are not shown, the parser should provide it, helps keep the file shorter

Chapter 2 - Telemetry ParameterTypes – Reference Diagram



Chapter 2 - ValidRange Check

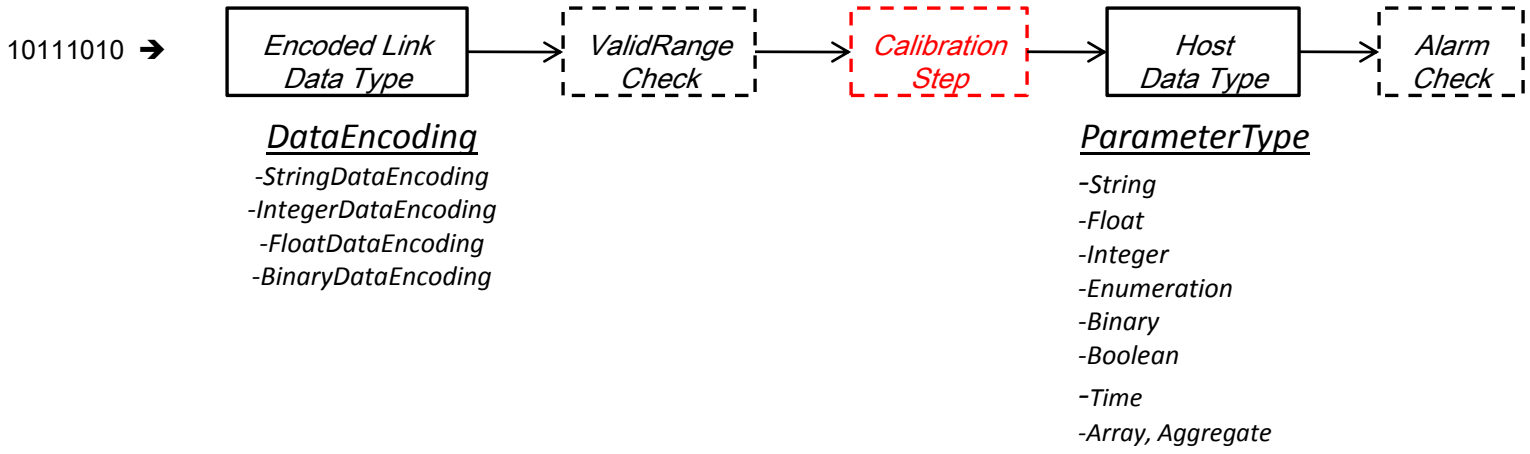
- ValidRange Check
  - IntegerParameterType and FloatParameterType only

<xtce:ValidRange minInclusive="0" maxInclusive="100"/>

Ignore for Telemetry side:
-IntegerParameterType/@validRangeAppliesToCalibrated=(**true** | false)
-FloatParameterType/@validRangeAppliesToCalibrated=(**true** | false)

In essence the ValidRange check for telemetry is assumed to be on the RAW DataEncoding values, the current version of XTCE is not clear on this.

Chapter 2 - Telemetry ParameterTypes – Reference Diagram



Chapter 2 - Calibrators

- Calibrators are defined in the DataEncoding area
- A DefaultCalibrator
  - Just one
- Or ContextCalibrators
  - Many
  - Conditional “Contexts”, user defined (e.g. Mission Phases)
- We’ll look at two common types
  - polynomial (PolynomialCalibrator)
  - line segment (SplineCalibrator)

Chapter 2 - Linear Calibrator


```
<xtce:SplineCalibrator>
   <!--Forward (regular) calibration   -->
   <xtce:SplinePoint  raw="1" calibrated="10"/>
   <xtce:SplinePoint  raw="2" calibrated="100"/>
   <xtce:SplinePoint  raw="3" calibrated="500"/>
</xtce:SplineCalibrator>
```

There's an optional @order attribute in SplinePoint – this is a typo in the Schema, ignore

But just to be confusing – there's an optional SplineCalirbator attribute @order which can be used to designate higher order splines. However there may need to be more info supplied (in Ancillary) to fully describe the details of your spline in the current version of XTCE

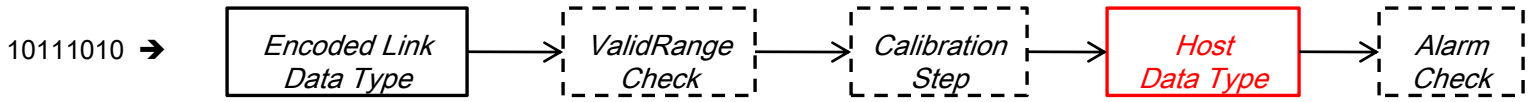
Chapter 2 - Polynomial Calibrator

The equation for the calibration is: $y = -0.0048x^2 + 1.4091x - 48.886$

```
<xtce:PolynomialCalibrator>
   <!--Forward (regular) calibration   -->
   <xtce:Term exponent="0" coefficient="-48.886"/>          – 48.886
   <xtce:Term exponent="1" coefficient="1.4091"/>            1.4091x
   <xtce:Term exponent="2" coefficient="-0.0048"/>          – 0.0048x²
</xtce:PolynomialCalibrator>
```

The number of terms is unlimited in the Schema you may wish to restrict this for your mission

Chapter 2 - Telemetry ParameterTypes – Reference Diagram



DataEncoding
-StringDataEncoding
-IntegerDataEncoding
-FloatDataEncoding
-BinaryDataEncoding

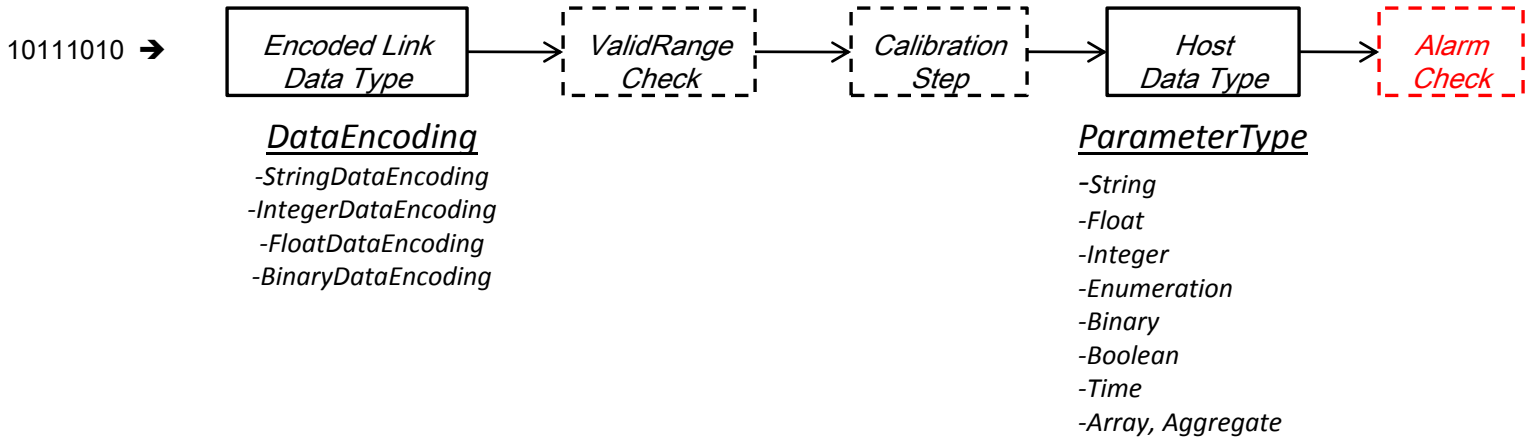
ParameterType:
String,
Enumerated,
Binary (catch all),
Integer
Float
Boolean
AbsoluteTime
RelativeTime
Array
Aggregrate (groups of other Parameters)

Chapter 2 – Likely ParameterType/DataEncoding Combinations

ParameterType	DataEncoding	Result
StringParameterType	StringDataEncoding	Unicode String encoded as either UTF-8 or UTF-16
EnumeratedParameterType	IntegerDataEncoding	LABEL, VALUE pairs
BinaryParameterType	BinaryParameterType	Blob data
IntegerParameterType	IntegerDataEncoding	• Integers of various well known formats (one/twos complement, etc…) • Calibrated/uncalibrated an option
FloatParameterType	FloatDataEncoding	•Floats of well known formats (IEEE/MIL1750A) • Calibrated/uncalibrated an optiona
FloatParameterType	IntegerDataEncoding	•Integer counts to units conversion • Calibrated
BooleanParameterType	IntegerDataEncoding	True/False
AbsoluteTimeParameterType	IntegerDataEncoding	•Time, same for RelativeTimeParameterType
Any of the above	BinaryDataEncoding	Format not describable with what is there
Aggregate or Array ParameterType	N/A	These NameReference other ParameterTypes

Others? May not make sense to some but not illegal!

Chapter 2 - Telemetry ParameterTypes – Reference Diagram



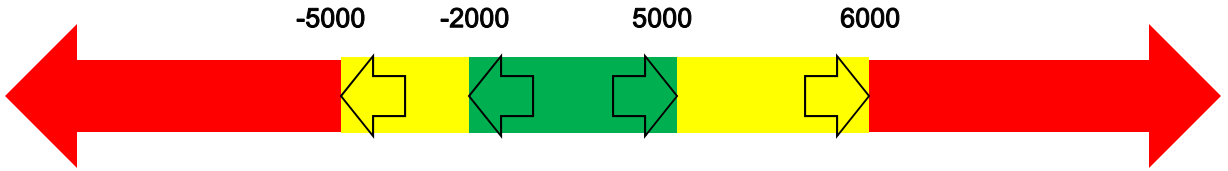
Chapter 2 - Alarms

- DefaultAlarm and ContextAlarms
  - Default is ... well the default
  - Contexts – user defined such as mission phase
- A variety of limit checks are available in XTCE and tuned to their ParameterType somewhat:
  - AlarmConditions: check a condition
  - StaticAlarmRanges: compare values against set of ranges
  - ChangeAlarms: Rate or Delta
  - Slight variations for String & Enum alarms

Chapter 2 – StaticAlarmRange Example

```
<xtce:StaticAlarmRanges>
        <xtce:WatchRange minInclusive="-2000.0" maxInclusive="5000.0"/>
        <xtce:SevereRange minInclusive="-5000.0" maxInclusive="6000.0"/>
</xtce:StaticAlarmRanges>
```

Real Ranges:
-Green: -2000 .0 > *x* < 5000.0 – implied
-Watch: *x* <= -2000.0 or *x* >= 5000.0
-Severe: *x* <= -5000.0 or *x* >= 6000.0

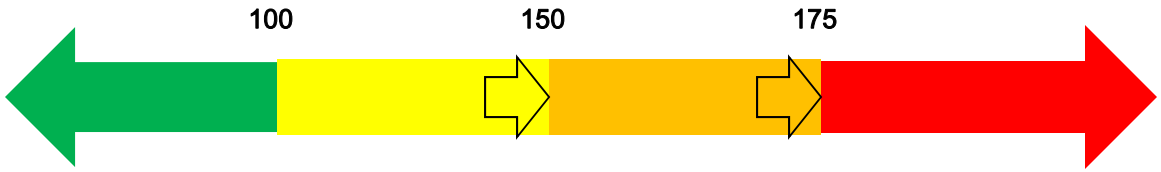


Note: Color designation is not part of XTCE, user dependent

Chapter 2 - AlarmConditions

- Set up conditions for up to five levels
  - The alarm “returns” the highest level triggered
  - Simple single conditions, multiple conditions, boolean expressions, custom algorithms

```
<xtce:AlarmConditions>
    <xtce:WarningAlarm>
        <xtce:Comparison parameterRef="pressureValue" value="100" comparisonOperator="&gt;"/>
    </xtce:WarningAlarm>
    <xtce:CriticalAlarm>
        <xtce:Comparison parameterRef="pressureValve" value="150" comparisonOperator="&gt;"/>
    </xtce:CriticalAlarm>
    <xtce:SevereAlarm>
      <xtce:Comparison parameterRef="pressureValve" value="175" comparisonOperator="&gt;"/>
    </xtce:SevereAlarm>
</xtce:AlarmConditions>
```



Chapter 2 - ChangeRateAlarms

- Similar to the others in terms of ranges but two choices in one element...

Rate of Change		**Delta Change**	
Attribute	Value	Attribute	Value
@changeType	changePerSecond	@changeType	changePerSample
@changeBasis	absoluteChange | **percentageChange**	@changeBasis	**absoluteChange** | percentageChange
@spanOfInterestInSamples	ignore	**@spanOfInterestInSamples**	1 or more
@spanOfInterestInSeconds	1 or more	@spaceOfInterestInSeconds	ignore

- Either a 1st derivative alarm that is either with respect to time or with respect to samples

IntegerParameterType

```
<xtce:IntegerParameterType signed="false" name="MySampleType">
  <xtce:UnitSet/>
  <xtce:IntegerDataEncoding sizeInBits="32"/>
  <xtce:DefaultAlarm>
    <xtce:StaticAlarmRanges>
      <xtce:WatchRange minInclusive="-2000.0" maxInclusive="5000.0"/>
      <xtce:WarningRange minInclusive="-5000.0" maxInclusive="6000.0"/>
    </xtce:StaticAlarmRanges>
  </xtce:DefaultAlarm>
</xtce:IntegerParameterType>
```

Chapter 2 - ParameterType Examples

FloatParameterType

```
<xtce:FloatParameterType sizeInBits="64" name="DecaySensorType">
  <xtce:UnitSet>
    <xtce:Unit description="Bq">Becquerel</xtce:Unit>
  </xtce:UnitSet>
  <xtce:IntegerDataEncoding sizeInBits="16" encoding="twosCompliment">
    <xtce:DefaultCalibrator>
      <xtce:SplineCalibrator>
        <xtce:SplinePoint raw="-32768.0" calibrated="0.0"/>
        <xtce:SplinePoint raw="0.0" calibrated="5.0"/>
        <xtce:SplinePoint raw="32767.0" calibrated="20.0"/>
      </xtce:SplineCalibrator>
    </xtce:DefaultCalibrator>
  </xtce:IntegerDataEncoding>
</xtce:FloatParameterType>
```


```
<xtce:SpaceSystem>
    <xtce:TelemetryMetaData>
      <xtce:ParameterTypeSet/>
      <xtce:ParameterSet>
    </xtce:TelemetryMetaData>
    <xtce:CommandMetaData/>
    <xtce:SpaceSystem/>
</xtce:SpaceSystem>
```

Chapter 2 – A Parameter has a ParameterType

- Parameter
  - Holds a name of a telemetry item -- "**BAV10089-B**"
  - And a NameRef – to a ParameterType
  - And an optional "ParameterProperties/@dataSource" attribute
    - Telemetered (default – cmd side ignore)
    - Derived, local, etc...
  - ParameterTypes may be shared by different Parameters...
  - An easier implementation approach is to have a one to one mapping between every Parameter and a ParameterType definition.

```
<xtce:ParameterTypeSet>
 <xtce:IntegerParameterType name="MyParameterType">
    <xtce:IntegerDataEncoding/>
  <xtce:IntegerParameterType/>
</xtce:ParameterTypeSet>
<xtce:ParameterSet>
    <xtce:Parameter name="BAV10089-B" parameterTypeRef="MyParameterType"/>
</xtce:ParameterSet>
```

Chapter 2 – Session or System Variables

- To define a variable supplied by the system
  - Simply leave off the DataEncoding in ParameterType
  - And then make Parameter that refers to it
  - The assumption is that somehow the system will supply the state of the variable
  - XTCE does not define how this is done, it's just a representation of it
  - Session variables often seem to be useful in a Comparison, where the information is held outside the all the descriptions in the file...

<xtce:StringParameterType name="HostNameType">

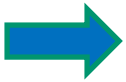
Chapter 3 – SequenceContainers

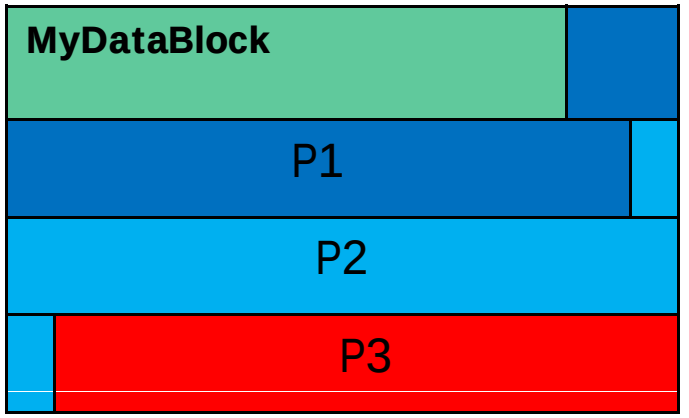

```
<xtce:SpaceSystem>
    <xtce:TelemetryMetaData>
        <xtce:ParameterTypeSet/>
        <xtce:ParameterSet/>
        <xtce:ContainerSet/>
    </xtce:TelemetryMetaData>
    <xtce:CommandMetaData/>
    <xtce:SpaceSystem/>
</xtce:SpaceSystem>
```

Chapter 3 - SequnceContainers

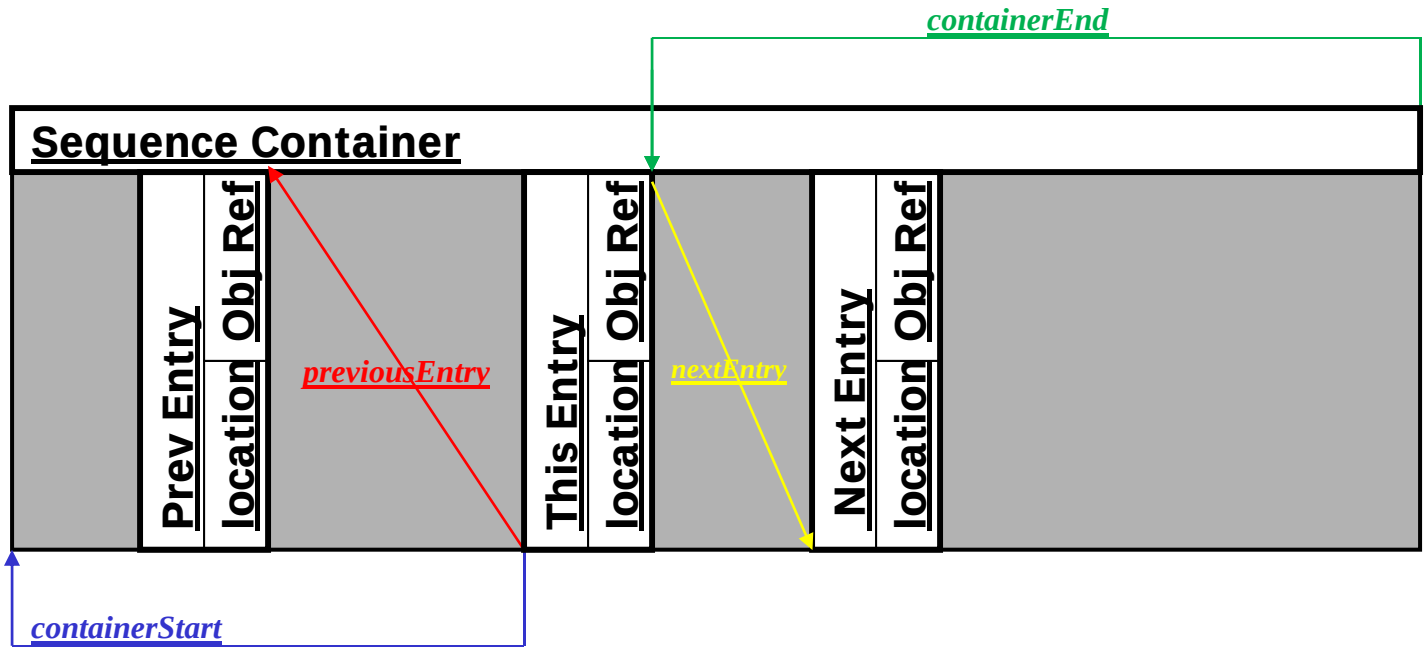
- Use to represent “memory layout” of Parameters (e.g. a packet or minor frame)
  - EntryList specifies content and layout:
    - Default: items are “packed” back to back
    - Width taken from ParameterType
      - (look up Parameter, then its ParameterType, then ParameterType’s Encoding, then the sizeInBits in the Encoding)
  - Element: <SequenceContainer>
    - Very general, may refer to other SeqContainers, Parameters

```
<SequenceContainer name="MyDataBlock">
  <EntryList>
    <ParameterRefEntry parameterRef="P1"/>
    <ParameterRefEntry parameterRef="P2"/>
    <ParameterRefEntry parameterRef="P3"/>
  </EntryList>
</SequenceContainer>
```





- Location of the entry is an integer value referenced from:
  - The end of the previous entry to the start of This Entry (**default**)
  - The start of the container to the start of This Entry (containerStart)
  - The end of the container to the end of This Entry (containerEnd)
  - The start of the next entry to the end? of this entry (nextEntry)



Chapter 3 - Containers – EntryList Options

- By default EntryList assumes items are “packed”
  - this is the simplest form and easiest to implement
- But there are several ways to modify an Entry
  - Location
  - Condition to include it
  - Repeats

```
<xtce:EntryList>
   <xtce:ParameterRefEntry parameterRef="Parameter1">
      <xtce:LocationInContainerInBits referenceLocation="containerStart">
         <xtce:FixedValue>0<xtce:FixedValue>
      </xtce:LocationInContainerInBits>
   </xtce:ParameterRefEntry>
   <xtce:ParameterRefEntry parameterRef="OptionalParameter">
     <xtce:IncludeCondition>
       <xtce:Comparison parameterRef="statusCheck" value="1"/>
     </xtce:IncludeCondition>
   </xtce:ParameterRefEntry>
</xtce:EntryList>
```

Chapter 3 - Simple Example

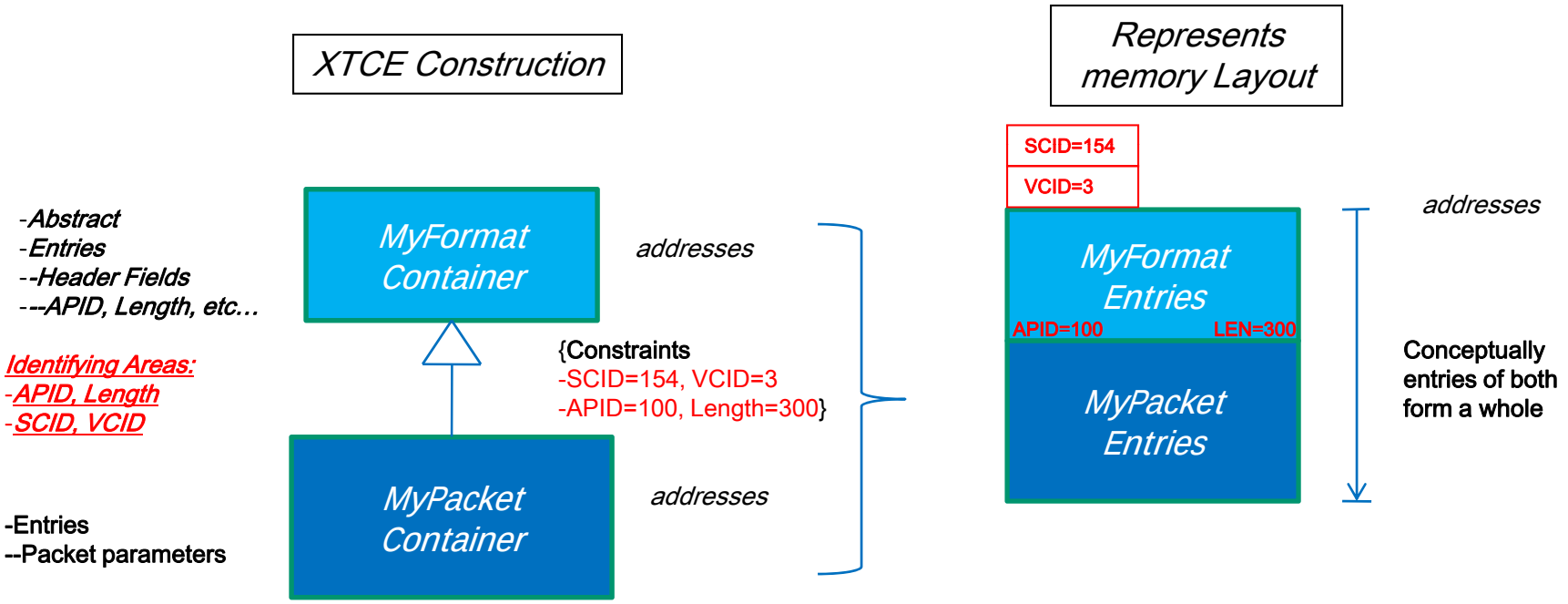

```
<xtce:ParameterTypeSet>
   <xtce:IntegerParameterType name="MyParameterType"/>
 </xtce:ParameterTypeSet>
 <xtce:ParameterSet>
   <xtce:Parameter name="MyParameter" parameterTypeRef="MyParameterType"/>
 </xtce:ParameterSet>
<xtce:SequenceContainer name="MyPacket">
   <xtce:EntryList>
      <xtce:ParameterRefEntry parameterRef="MyParameter"/>
      <xtce:ContainerRefEntry containerRef="OtherContainer"/>
   <xtce:EntryList>
</xtce:SequenceContainer>
<xtce:SequenceContainer name="OtherContainer">
   <xtce:EntryList>
      <xtce:ParameterRefEntry parameterRef="Parameter2"/>
   <xtce:EntryList>
</xtce:SequenceContainer>
```

Chapter 3 – Container Inheritance

- Use Container Inheritance to add identifying "keys" to a full description
  - keys are identifying locations in the bit-stream, like CCSDS VCID, APID, packet length or format id, minor frame number
  - BaseContainer child element
    - Supply the Parent or "Super" Container Ref
    - Supply "Constraints" – a set of conditions that must be true for this description
      - Conditions are usually Parameters in the Parent Container
      - The condition Parameters are probably IDENTIFYING areas of a packet/minor frame:
        - » Ex. CCSDS Land:
        - » APID
        - » Packet Length
        - » SCID, VCID (note: these are not in packet header – try session variables!)
- Somewhat like class inheritance but more of a construction technique
  - You do get to say this container IS A that container
  - But instead of method overloading the EntryList of the parent is added to the child's
  - And constraints (RestrictionCriteria) are defined, usually interpreted as identifying keys in the bit-stream

Chapter 3 – Container Inheritance in Pictures

MyPacket



The constraints allow one to match incoming bits with these descriptions – whether literally or conceptually. Identifying areas and value for that particular description.

Chapter 3 -Telemetry Packet Description -

MyPacket

The XTCE constructions:

```
<xtce:SequenceContainer name="MyFormat" abstract="true">
  <xtce:EntryList>
    <xtce:ParameterEntryRef parameterRef="APID>
    <xtce:ParameterEntryRef parameterRef="LEN>
  </xtce:EntryList>
</xtce:SequenceContainer>

<xtce:SequenceContainer name="MyPacket>
  <xtce:EntryList>
    <xtce:ParameterEntryRef parameterRef="P1">
    <xtce:ParameterEntryRef parameterRef="P2">
    <xtce:ParameterEntryRef parameterRef="P3">
  </xtce:EntryList>
  <xtce:BaseContainer containerRef="MyFormatPacket">
  <xtce:RestrictionCriteria>
    <xtce:ComparisonList>
      <xtce:Comparison parameterRef="APID" value="100"/>
      <xtce:Comparison parameterRef="LEN" value="300"/>
      <xtce:Comparison parameterRef="SCID" value="154"/>
      <xtce:Comparison parameterRef="VCID" value="3"/>
</xtce:RestrictionCriteria>
  </xtce:BaseContainer>
</xtce:SequenceContainer>
```

MyFormat IS A SequenceContainer

MyPacket IS A MyFormat

Identifying Constraints

Conceptual Result

```
<MyPacket>
 <Entries>
   <APID/>
   <LEN/>
   <P1/>
   <P2/>
   <P3/>
   <When>
     <APID==100/>
     <LEN==300/>
     <SCID==154/>
     <VCID==3/>
   </When>
 </Entries>
</MyPacket>
```

NOTE: SCID and VCID are Session Variables, supplied by the System in this construction – NOT SHOWN


```
<xtce:SpaceSystem>
    <xtce:TelemetryMetaData>
      <xtce:ParameterTypeSet/>
      <xtce:ParameterSet/>
      <xtce:ContainerSet/>
    </xtce:TelemetryMetaData>
    <xtce:CommandMetaData/>
    <xtce:SpaceSystem/>
</xtce:SpaceSystem>
```

Chapter 4 – CommandMetaData

Command Descriptions

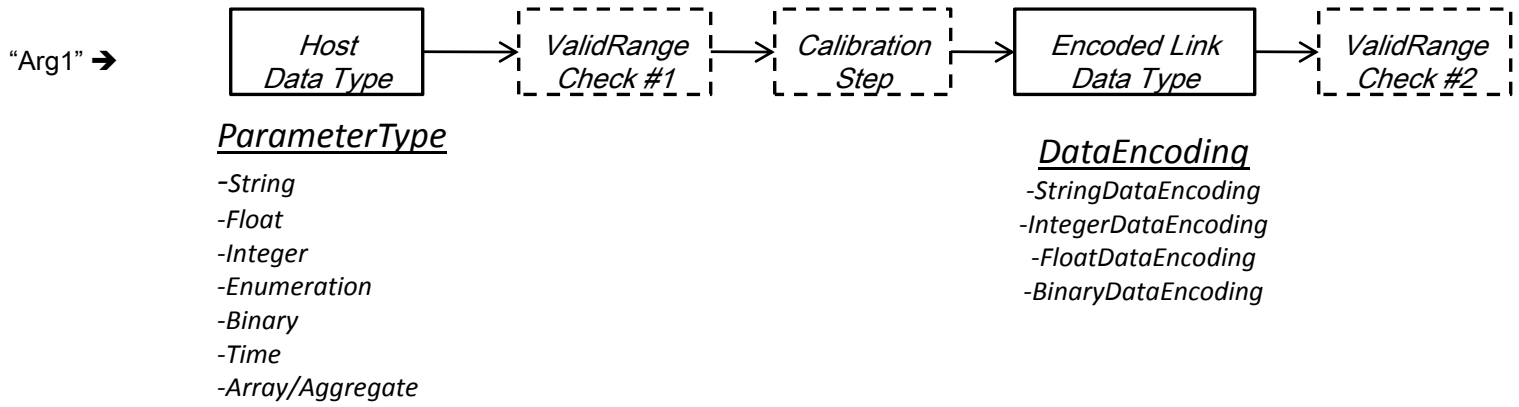
- Describes
  - Command and args
  - The Command Packets (or minor frames)
  - Command Parameters
  - Various other aspects of Commanding – verification, etc...
- Shares some schema-types with TelemetryMetaData
  - ParameterSet and ParameterTypeSet are exactly the same
  - ArgumentTypeSet is similar to ParameterTypeSet
  - CommandContainerSet is the same as ContainerSet
- This section then will focus on the differences between the telemetry and command side in XTCE
  - Many construction similar to telemetry but difference highlighted
  - Even same XML construction may have a slightly different meaning

Chapter 4 - Major CommandMetaData Elements


```
<xtce:SpaceSystem>
  <xtce:TelemetryMetaData>
      <xtce:ParameterTypeSet/>
      <xtce:ParameterSet/>
      <xtce:ContainerSet/>
  </xtce:TelemetryMetaData>
  <xtce:CommandMetaData>
      <xtce:ParameterTypeSet/>
      <xtce:ParameterSet/>
      <xtce:ArgumentTypeSet/>
      <xtce:MetaCommandSet/>
      <xtce:CommandContainerSet/>
    </xtce:CommandMetaData>
</xtce:SpaceSystem>
```

Chapter 4 - Command ParameterTypes and ArgumentTypes

- Command ParameterTypes use the same Schema-types as Telemetry ParameterTypes
  - They look identical
- ArgumentTypes are similar in construction to Telemetry ParmeterType as well
  - But they lack alarms (Command ParameterType probably should too)
- Relationship of child-elements in Type has slightly different meaning:
  - Order is interpreted differently than telemetry – calibrators are "reverse" (aka raw to units)
  - Valid range check may occur two places depending on @validRangeAppliesToCalibrated value



- *Generally the various accepted combinations Type/DataEncoding are the same for Telemetry ParameterTypes*
- *ValidRange is slightly different, and no alarms should be specified*

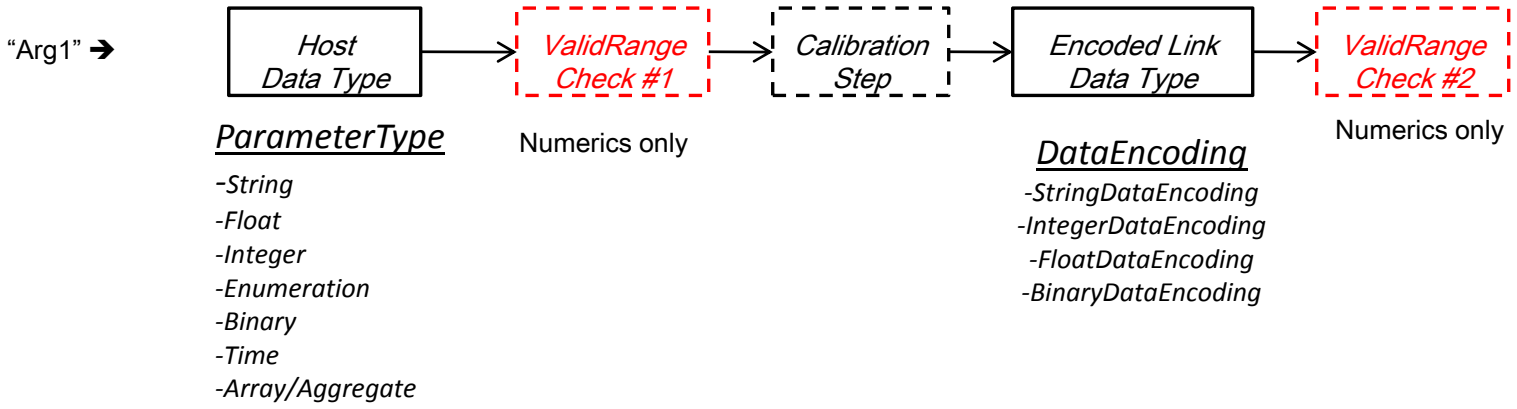
Chapter 4 - CommandMetaData Parameters and Arguments


```
<xtce:SpaceSystem>
  <xtce:TelemetryMetaData>
      <xtce:ParameterTypeSet/>
      <xtce:ParameterSet/>
      <xtce:ContainerSet/>
  </xtce:TelemetryMetaData>
  <xtce:CommandMetaData>
      <xtce:ParameterTypeSet/>
      <xtce:ParameterSet/>
      <xtce:ArgumentTypeSet/>
      <xtce:MetaCommandSet>
          <xtce:MetaCommand>
              <xtce:ArgumentList/>
          </xtce:MetaCommand>
      </xtce:MetaCommandSet>
      <xtce:CommandContainerSet/>
    </xtce:CommandMetaData>
</xtce:SpaceSystem>
```

Chapter 4 - Command Parameters vs Arguments

- Command Parameters and Arguments are very similar but have a different meaning:
  - Command parameter values would be supplied by the system or their value specified in a RestrictionCriteria
  - Arguments are supplied by the user
  - Example:
    - A checksum in a command packet is probably a good candidate for command parameter, the user will never see it or even know it is there
  - Arguments are defined local to a MetaCommand
- Other than this the two are not that different
  - Arguments Ref ArgumentTypes, Parameters Ref ParameterTypes
  - And they are constructed from closely related Schema Types

Chapter 4 - Context Diagram



Chapter 4 - ValidRange in Commanding

- ValidRange in Commanding
  - Unlike Telemetry -- can be set to apply BEFORE the command or AFTER the command (but not both)
    - BEFORE
      - leave attribute validRangeAppliesToCalibrated true (default)
    - AFTER
      - set attribute validRangeAppliesToCalibrated to false

Chapter 4 – CommandMetaData Describing Commands


```
<xtce:SpaceSystem>
  <xtce:TelemetryMetaData>
     <xtce:ParameterTypeSet/>
     <xtce:ParameterSet/>
     <xtce:ContainerSet/>
  </xtce:TelemetryMetaData>
  <xtce:CommandMetaData>
     <xtce:ParameterTypeSet/>
     <xtce:ParameterSet/>
     <xtce:ArgumentTypeSet/>
     <xtce:MetaCommandSet>
          <xtce:MetaCommand>
               <xtce:ArgumentList/>
               <xtce:CommandContainer>
          </xtce:MetaCommand>
     </xtce:MetaCommandSet>
     <xtce:CommandContainerSet/>
   </xtce:CommandMetaData>
</xtce:SpaceSystem>
```

MetaCommand/CommandContainer has a LOCAL CommandContainer for Commands

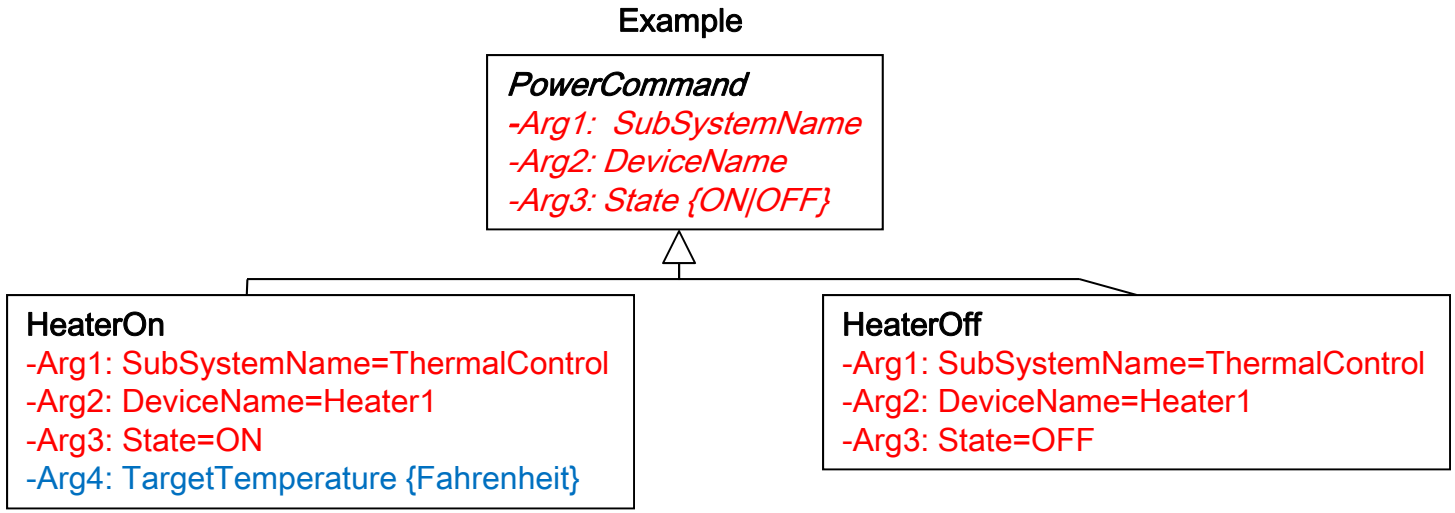
- Build Command Packets here!

Chapter 4 - Describing Commands and Command Packets w/ the MetaCommand Element

- Use the MetaCommand element to describe Commands
  - Supply Command Arguments
  - A local Container for their Packets (or minor frame)
  - Add any Verifiers, Interlock, Priority, various constraints, side effects
  - Optional use **MetaCommand Inheritance** to build up the Command
    - Similar to Container Inheritance
- Use its local MetaCommand/CommandContainer to build its Packet (or Minor Frame)
  - Reference Command Parameters
  - Reference Command Arguments
  - And it may Reference other CommandContainers (CommandContainerSet)
  - Also has Inheritance Mechanism similar to Container Inheritance

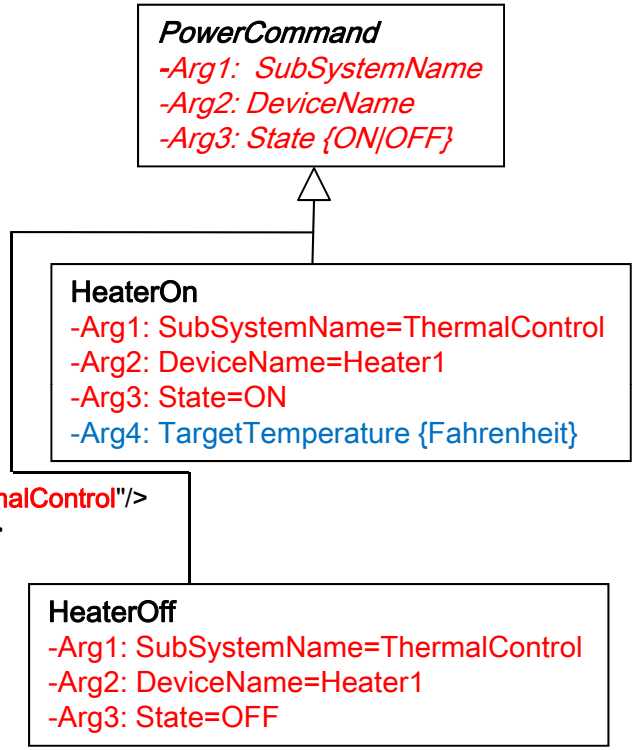
Chapter 4 – Command Inheritance

- A MetaCommand may EXTEND another
  - One Command may EXTEND another
  - BaseMetaCommand
    - Give the Ref of the MetaCommand being extended
- The extending command can add arguments
  - It gets any Parent arguments and adds its own
- Or it can set arguments in the Parent's Command
  - MetaCommand/BaseMetaCommand/ArgumentAssigmentList



Chapter 4 – Command Inheritance in XTCE

```
<xtce:MetaCommand name="PowerCommand" abstract="true">
      <xtce:ArgumentList>
<xtce:Argument name="SubSystemName" argumentTypeRef="SubSysEnumType"/>
<xtce:Argument name="DeviceName" argumentTypeRef="DeviceEnumType"/>
<xtce:Argument name="State" argumentTypeRef="StateEnumType"/>
      </xtce:ArgumentList>
      <xtce:CommandContainer/>  <!– details left out -->
</xtce:MetaCommand>
<xtce:MetaCommand name="HeaterOn">
         <xtce:ArgumentList>
<Argument name="TargetTemperature argumentTypeRef="FloatType"/>
        </xtce:ArgumentList>
        <xtce:BaseMetaCommand metaCommandRef="PowerCommand">
              <xtce:ArgumentAssignmentList>
                  <xtce:ArgumentAssignment argumentName="SubSystemName" argumentValue="ThermalControl"/>
                  <xtce:ArgumentAssignment argumentName="DeviceName" argumentValue="Heater1"/>
                  <xtce:ArgumentAssignment argumentName="State" argumentValue="ON"/>
              </xtce:ArgumentAssignmentList>
       </xtce:BaseMetaCommand>
       <xtce:CommandContainer/>
</xtce:MetaCommand>
<xtce:MetaCommand name="HeaterOff">
        <xtce:BaseMetaCommand metaCommandRef="PowerCommand"/>
              <xtce:ArgumentAssignmentList>
<xtce:ArgumentAssignment argumentName="SubSystemName" argumentValue="ThermalControl"/>
<xtce:ArgumentAssignment argumentName="DeviceName" argumentValue="Heater1"/>
<xtce:ArgumentAssignment argumentName="State" argumentValue="OFF"/>
              </xtce:ArgumentAssignmentList>
         </xtce:BaseMetaCommand>
         <xtce:CommandContainer/>
</xtce:MetaCommand>
```



Chapter 4 – MetaCommand's CommandContainer

- MetaCommand/CommandContainer
  - Similar to the other Containers but not exactly the same
    - It has a FixedValueEntry to hard code a value
    - It has an ArgumentRefEntry for Arguments
      - (and ArrayArgument and AggrateArgument)
    - It has a BaseContainer but its RestrictionCriteria is optional
    - It has no abstract attribute either!
- Use it to build the Packet or Minor Frame associated with the MetaCommand, the EntryList can have:
  - ParameterRefEntry and its various forms
  - ArgumentRefEntry, ArgumentArrayRefEntry, ArgumentAggregateRefEntry
  - FixeValueEntry
  - ContainerRefEntry and its various forms
    - Don't Ref another MetaCommand/Container
    - Instead Ref in CommandContainerSet!

Chapter 4 – MetaCommand's CommandContainer in XTCE

```
<xtce:MetaCommand name="PowerCommand" abstract="true">
      <xtce:ArgumentList>
<xtce:Argument name="SubSystemName" argumentTypeRef="SubSysEnumType"/>
<xtce:Argument name="DeviceName" argumentTypeRef="DeviceEnumType"/>
<xtce:Argument name="State" argumentTypeRef="StateEnumType"/>
      </xtce:ArgumentList>
      <xtce:CommandContainer name="PowerCommandPacket">
            <xtce:EntryList>
              <xtce:FixedValueEntry binaryValue="00a0" sizeInBits="16"/>
              <xtce:ParameterRefEntry parameterRef="CheckSum"/>
              <xtce:ArgumentRefEntry argumentRef="SubSystemName"/>
              <xtce:ArgumentRefEntry argumentRef="DeviceName"/>
              <xtce:ArgumentRefEntry argumentRef="State"/>
            </xtce:EntryList>
     </xtce:CommandContainer>
</xtce:MetaCommand>
```

Packet name

"OpCode"

Check sum, calculated by the system, inserted here before uplink

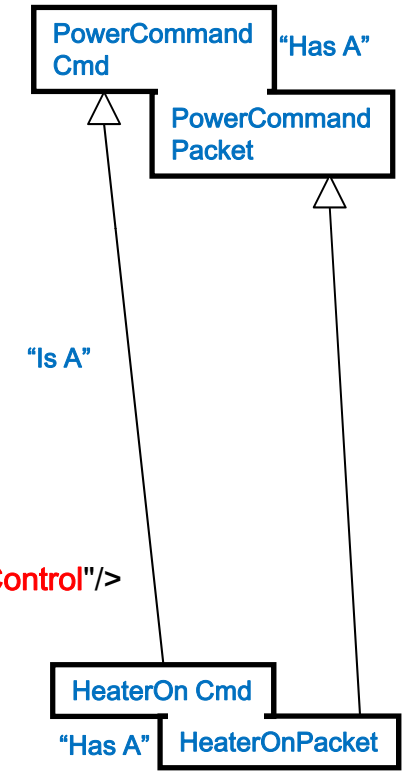
The arguments – order does not have to match ArgumentList but they should all be here…

Chapter 4 - MetaCommand CommandContainer Inheritance

- A MetaCommand/CommandContainer may EXTEND another
  - This typically occurs if you are using MetaCommand Inheritance
  - Similar to Container inheritance in the rest of XTCE
  - But … RestrictionCriteria is optional
  - Must EXPLICITLY set BaseContainer (XTCE's syntax)
  - That is – the MetaCommand/CommandContainer/BaseContainer to the Parent's MetaCommand/CommandContainer
    - (easier to visualize than explain)

Chapter 4 - MetaCommand CommandContainer Inheritance in XTCE

```
<xtce:MetaCommand name="PowerCommand" abstract="true">
    <xtce:ArgumentList>
     <xtce:Argument name="SubSystemName" argumentTypeRef="SubSysEnumType"/>
     <xtce:Argument name="DeviceName" argumentTypeRef="DeviceEnumType"/>
     <xtce:Argument name="State" argumentTypeRef="StateEnumType"/>
    </xtce:ArgumentList>
    <xtce:CommandContainer name="PowerCommandPacket">
        <xtce:EntryList>
            <xtce:FixedValueEntry binaryValue="00a0" sizeInBits="16"/> <!– opcode -->
            <xtce:ArgumentRefEntry SubSystemName, DeviceName, State/> <!– illustrative -->
        </xtce:EntryList>
    </xtce:CommandContainer>
</xtce:MetaCommand>
<xtce:MetaCommand name="HeaterOn">
      <xtce:ArgumentList>
        <Argument name="TargetTemperature" argumentTypeRef="FloatType"/>
      </xtce:ArgumentList>
      <xtce:BaseMetaCommand metaCommandRef="PowerCommand">
        <xtce:ArgumentAssignmentList>
          <xtce:ArgumentAssignment argumentName="SubSystemName" argumentValue="ThermalControl"/>
          <xtce:ArgumentAssignment argumentName="DeviceName" argumentValue="Heater1"/>
          <xtce:ArgumentAssignment argumentName="State" argumentValue="ON"/>
        </xtce:ArgumentAssignmentList>
      </xtce:BaseMetaCommand>
      <xtce:CommandContainer name="HeaterOnPacket">
          <xtce:EntryList/>
          <xtce:BaseContainer containerRef="PowerCommandPacket"/>
      </xtce:CommandContainer>
</xtce:MetaCommand>
```



Chapter 4 - MetaCommand CommandContainer Inheritance in XTCE w/RestrictionCriteria

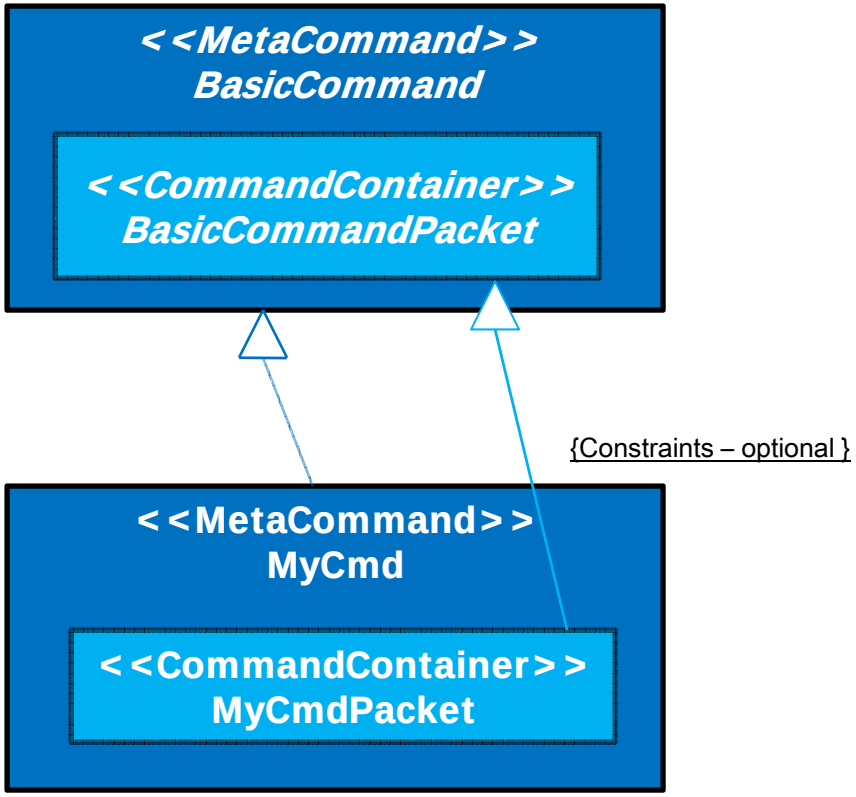

```
<xtce:MetaCommand name="PowerCommand" abstract="true">
    <xtce:ArgumentList/> <!-- args removed for illustrative purposes -->
    <xtce:CommandContainer name="PowerCommandPacket">
        <xtce:EntryList>
 <xtce:ParameterRefEntry parameterRef="OpCode"/>
            <!-- other entries not shown -->
        </xtce:EntryList>
    </xtce:CommandContainer>
</xtce:MetaCommand>

<xtce:MetaCommand name="HeaterOn">
        <xtce:ArgumentList>
<Argument name="TemperatureCutoff" argumentTypeRef="FloatType"/>
        </xtce:ArgumentList>
        <xtce:BaseMetaCommand metaCommandRef="PowerCommand">
          <xtce:ArgumentAssignmentList/> <!-- arg assignments removed for illustrative purposes -->
       </xtce:BaseMetaCommand>
       <xtce:CommandContainer name="HeaterOnPacket">
            <xtce:EntryList/> <!-- other entries not shown -->
            <xtce:BaseContainer containerRef="PowerCommandPacket">
                <xtce:RestrictionCriteria>
                  <xtce:Comparison parameterRef="OpCode" value="00a0"/>
                </xtce:RestrictionCriteria>
            </xtce:BaseContainer>
       </xtce:CommandContainer>
</xtce:MetaCommand>
```

System supplies the value for OpCode…

But checks the value here in this constraint… ?Opcode == 0xa0?

Simple constraints of the style "parameter == value" could be automatically processed to essentially inform the system the values that MUST be supplied in the named parameters. You are encouraged to stick with that form for this reason but this is not required.

Chapter 4 – MetaCommand and MetaCommand/CommandContainer Inheritance Summary



A MetaCommand may EXTEND another: MyCmd extends BasicCommand above
Their local CommandContainers extend each other: MyCmdPacket extends BasicCommandPacket

- *Constraints are optional here*

Chapter 4 - Commanding:Other Items

- Other commanding items in MetaCommand:
  - TransmissionConstraint - <TransmissionConstraintList>
    - Check cmd can be run
      - Implied blockage if constraints fail
  - Significance - <DefaultSignificance>, <ContextSignificanceList>
    - Permission/confirmations
  - Interlock
    - Constraints on the next command
  - Verification -- <VerifierSet>
    - Variety of command verification by stages (Command Complete)
  - Side Effects – <ParameterToSetList>
    - Side effects after command sent
      - E.g. Command Counter, etc...
  - Suspend Alarms until... <ParameterToSuspendAlarmsSet>
    - Suspend named parameters while cmd takes effect

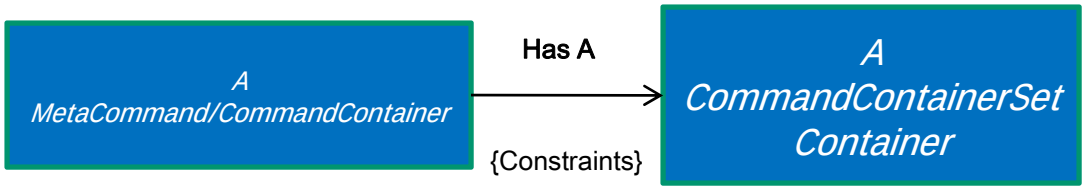
Chapter 4 – CommandContainerSet


```
<xtce:SpaceSystem>
  <xtce:TelemetryMetaData>
      <xtce:ParameterTypeSet/>
      <xtce:ParameterSet/>
      <xtce:ContainerSet/>
  </xtce:TelemetryMetaData>
  <xtce:CommandMetaData>
      <xtce:ParameterTypeSet/>
      <xtce:ParameterSet/>
      <xtce:ArgumentTypeSet/>
      <xtce:MetaCommandSet>
          <xtce:MetaCommand>
              <xtce:ArgumentList/>
              <xtce:CommandContainer>
          </xtce:MetaCommand>
      </xtce:MetaCommandSet>
      <xtce:CommandContainerSet/>
    </xtce:CommandMetaData>
</xtce:SpaceSystem>
```

CommandContainerSet is NOT the same thing as MetaCommand/CommandContainer

Chapter 4 – CommandContainerSet

- Built using the same XTCE Schema-types as ContainerSet in TelemetryMetaData…
- Generally meant to be used as the “pieces/parts” of MetaCommand/CommandContainers
  - Chunks of command parameters that repeat or reused by various commands
- A “has a” relationship



For example – perhaps a Command Packet has an optional secondary header
- Constraint: Secondary Header Flag must be 1

Chapter 5 – The Forgotten Elements

- Lesser used major elements include:
  - TelemetryMetaData/MessageSet
  - StreamSet
  - AlgorithmSet
  - ServiceSet

Chapter 5 - MessageSet

- Deprecated...
- Essentially an alternative way to build a Packet or Minor Frame...
  - A purely “HAS A” relationship to Containers
  - Retained from an earlier version of XTCE

Chapter 5 - StreamSet – Streams

- A set of elements to describe aspects of “bit streams”
  - Possibly could be used for portions of frame description, etc...
- May be included in containers directly
  - Perhaps a special “sub-stream” in a container
- CCSDS – not quite enough elements to FULLY describe the “CCSDS stack” from the frame-sync to packets
  - Lacks RS encoding info for example (although one could use the ‘CustomStream’)
  - Generally focus on packet descriptions and leave the FEP for others
- FixedFrameStreams or VariableFrameStreams reference a top level abstract container
    - SequenceContainer identification techniques will determine the specific container
- Or links to a service
  - Services are abstractions, perhaps a service has certain streams within it...

Chapter 5 – AlgorithmSet

- Used to describe algorithms used in your tlm/cmd processing
  - Many options – from actually including code text, to simply the name of a function...
- Two forms:
  - Custom: functions, methods, procedures...
  - Math: equations using postfix notation...
- Usually define a 'trigger' that indicates when the algorithm should be invoked

Chapter 5 - Services

- An abstraction – groups “logically like” Containers
- For example:
  - Suppose your mission splits its functionality across multiple CCSDS Virtual Channels
  - Each Channel may represent a “service”
  - A Service element could be defined and the list of Containers for each APID in the service (VCID) added
    - Options:
      - point to the “super container” per channel
      - Point to each “final” packet container
      - Point to all the containers making up packets on those channels...
      - Etc...?
  - Totally user defined...

Chapter 6 - Forms of XTCE Implementations

- There are three ways XTCE has been suggested to be used:
  - #1 – the exchange form, as it was originally was intended
  - #2 – the central repository form
    - A central information database (repository) scenario
      - where it and perhaps other form of info are process in a variety of ways to produce “products” such as:
        - Flight software constructs
        - TLM & CMD databases
        - Documentation
  - #3 – the native form, in which it becomes directly part of a tool-chain
- The first form is by far and away the most popular
  - The second form has been proposed and explored by at least two missions
  - The third form has never been done
- In terms of the first form, it has been prototyped by at least the following:
  - NASA-GSFC, and various other NASA centers
  - ESA & Norwegian contractors for ESA
  - French Space Agency, Russian Space Agency, Germany Space Agency, EDS
  - ORS (USAF)
- It has been used in one mission for DARPA
  - Orbital Express (two satellites)
- It has two vendors (see GovSat for more coming online this year) – Harris OS/Comet, GMV Archiva

Chapter 6 – Implementing XTCE

How to a build a very general XTCE parser

- Step #1
  - Validate using XML validation against the official XTCE1.1 Schema
  - Bring XML into internal program "model" of class objects
- Step #2
  - Check for duplicate names (@names) by major element in all SpaceSystem, note that Telemetry and Command side often form one "namespace" in XTCE for this purpose
- Step #3
  - Resolve all NameReferences so that the actual item being referred to is an object
  - Note any "dangling" Refs (Refs that go to no location)
  - Check that Refs are by "Type" correctly, for example a ParameterRef to a ParameterType is indeed to a ParameterType, etc…
  - Check for loops in the Refs
  - See XTCE appinfo element (in the schema text), has additional validity checks that cannot be checked by the parser; written in human readable form
- Step #4
  - Implement XTCE's inheritance model for types, containers and commands (hold inheritance relationships in some easy to traverse data structure that tracks the parent and child relationships)
- Step #5
  - Match up any enumerations and enumeration alarms (verify the label specified is in the enumeration definition), match any initialValue with labels defined (doing this may mean looking in the parent type if there is one)
- You are now ready to process the information in a variety of ways … OR

Chapter 6 – Implementing XTCE 2

- A simplified XTCE parser can be implemented by outlawing the portions of XTCE that your mission is just not going to use AND the forms that are hard to implement
- The biggest areas of difficulty for most are generalized NameReferences, SpaceSystem trees, and any form of inheritance
- Certain concrete steps can make your job much easier:
- Step #1
  - Removing the SpaceSystem tree hierarchy immediately simplifies the NameReferences as only the simplest form of NameReference is necessary – just the name of the item is sufficient
  - This implies that the @name things are unique per file at the very least
- Step #2
  - Limit inheritance by selecting “patterns” for commands and tlm that will describe all your packets or minor frames
  - Often only one or two levels of inheritance is needed to describe all your content, headers and so forth can be placed in parent containers that don’t necessarily have even be processed by your software, as you may well know what the content should be…
- These two steps alone can vastly simplify an XTCE implementation, while more general implementations are desirable, the initiate may find the simpler approach is either entirely enough for the needs or at least a good place to start

Chapter 6 - Steps to Successfully Using XTCE in Exchange

- Step #1 - Enforce the use of XTCE within project
- Step #2 – Capture in a “rule book” TLM/CMD aspects for your mission:
  - Naming conventions
  - TLM/CMD format (“CCSDS Packet”, etc…)
  - Supported Data Types in TLM/CMD (MIL-1750A, etc…)
  - Time codes
  - Hardware dependencies (bit/byte order)
  - Limits, calibrators and phasing features (3 levels, 5 term polys, 4 phases)
  - Commanding aspects such validation and so forth…
- Step #3 – Map rules to XTCE and develop “XTCE rule validation tool” to enforce them
  - must parse against XTCE 1.1 Schema AND meet rules
- Step #4 – Distribute rule guide and tool within project
- Step #5 – Import/Export developed to/from each team member’s format following rules
  - Check any files with the “rule tool”

⇒ A lot of this is upfront planning the customer (probably) to do…

⇒ Its benefits accrue over time and across projects…

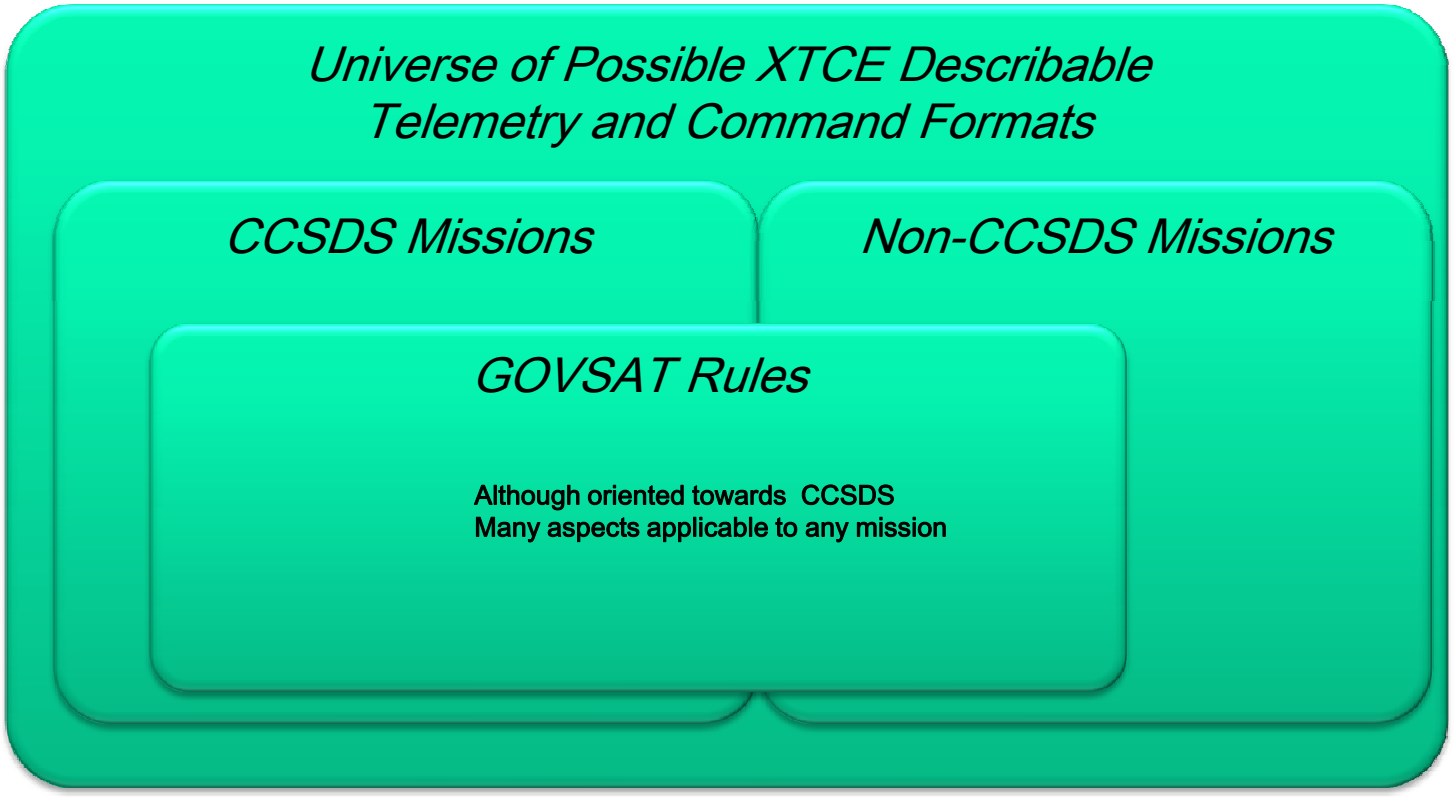
⇒ OR…

Chapter 7 - GovSat

- The previous discussion touched on XTCE's more complex areas
  - While nothing forces anyone to implement a full XTCE parser
  - Many have criticized some of its features as being overly complex, ambiguous, and too generalized
    - But this ignores the fact that each TLM/CMD community actually greatly differs in the details in many ways
- In addition, having each mission from scratch build XTCE rules guides specific to their mission is not ideal
- Options?
  - limit the way XTCE can used for a certain community (not just a mission)
  - get broad agreement among various agencies in that community on these limiting rules
  - get the vendors that service that community onboard
  - (implied) limit aspects of tlm/cmd architecture for space system

  ➔ GovSat

Chapter 7 - GovSat



Chapter 7 – GovSat

- Agencies:
  - NASA/GSFC, ORS, Airforce, NRO
  - Vendors or tools (agency):
    - ASIST, ITOS, SIMMS (GSFC)
    - SRA International/RIMMS (AF)
    - Harris/OS-Comet (AF, NRO)
- Specification:
  - XTCE 1.1 – subsets XTCE using rules, but does not create a new schema
    - All GovSat files should validate with XTCE1.1
  - Table of disallowed or allowed element and attributes
  - Some allowed attributes are further restricted
    - Ex. Name length up to 64 chars
  - Pattern for Commands and Packets
  - Description Report
  - Example DB
- Version:
  - 1.0 Draft Revision available now!
- Timeline:
  - Tool vendors complete in Feb to March of 2010
  - Demo testing to commence in April, finalization of final 1.0 release by May to end of FY10

Chapter 7 – GovSat Specifics (Draft 1.0)

- Of the ~900 or so XTCE elements and attributes
  - (including elements and attributes with repeat or a re-used in the schema)
  - About ~180 are completely disallowed
  - Of the remaining allowed elements, at least 20 are further restricted
- Restricts manner in which packets and commands are described using Containers and MetaCommand, and inheritance
  - Supports “GSFC-style” usage of CCSDS packet specs only
- Other Major Items:
  - Lists specific set of ParameterTypes that are supported
    - Simplifies Time related ParameterTypes
  - Restricts calibrators and alarms
  - Restricts SpaceSystems
  - Restrict NameReferences

Chapter 7 – GovSat Restricts SpaceSystems

- Only one SpaceSystem is legal
  - (thus Child SpaceSystems are illegal)
- Legal (direct) Child elements of SpaceSystem
  - Header element required but details left to mission
  - Required: AncillaryData “FileType”, “GovSat”
  - TelemetryMetaData
  - CommandMetaData

Chapter 7 – GovSat Restricts NameReferences

- GovSat only allows one form:
  - Unqualified -- just the name of the item of interest
    - no path information
- Because there is only one SpaceSystem
  - @name items are global to document by “element type”
- Taken together, the SpaceSystem restriction and @name globals:
  - Vastly simpler XTCE implementation

Chapter 7 – GovSat Restricts Inheritance

- Even with one SpaceSystem and unqualified NameReferences
  - General XTCE inheritance still hard to implement
- Instead GovSat restricts XTCE inheritance to two specific patterns
  - TLM Packet Pattern
  - CMD and CMD Packet Pattern
- There is an assumption that GovSat packets are:
  - CCSDS packets
  - And are used as NASA-GSFC missions use them
    - TLM secondary header is a time stamp on every packet
    - CMD side has no time stamp
    - Spacecraft ID, Virtual Channel ID, Application ID and packet length used to identify packets
    - And AppID is unique and packet may appear on more than one VCID
  - (Not all CCSDS missions in fact use CCSDS packets as GSFC uses them)

Chapter 7 – Telemetry Packets Pattern

- CCSDSPacket Container (one)
  - Root for all containers
  - Contains the CCSDS fields and nothing else
- Then CCSDSTelemetryPacket (one) extends CCSDSPacket
  - Adds no entries
  - Simply adds Restrictions (constraints) for telemetry flag check only
- (supplied processing software COULD ignore these constructs entirely as they may represent baseline info that is simply a given)
- Then each mission packet extends CCSDSTelemetryPacket
  - Supplies SCID, VCID, APID, packet length in Restriction Criteria
  - First entry is always time stamp
  - Container body is then filled in for the packet
  - These containers are considered final and may not be extended
- Overall a very simple pattern that requires minimal implementation

Chapter 7 – GovSat Telemetry Packet Eample

```
<xtce:SequenceContainer name="CAM0NumImagersReplyPacket">
  <xtce:EntryList>
    <xtce:ParameterRefEntry parameterRef="TimeStamp">
      <xtce:IncludeCondition>
      <xtce:Comparison value="1" parameterRef="CCSDSSecH"/>
      </xtce:IncludeCondition>
    </xtce:ParameterRefEntry>
    <xtce:ParameterRefEntry parameterRef="CAM0NumImagers"/>
  </xtce:EntryList>
  <xtce:BaseContainer containerRef="CCSDSTelemetryPacket">
    <xtce:RestrictionCriteria>
      <xtce:ComparisonList>
        <xtce:Comparison value="42" parameterRef="CCSDSSCID"/>
        <xtce:Comparison value="0" parameterRef="CCSDSVCID"/>
        <xtce:Comparison value="2" parameterRef="CCSDSAPID"/>
        <xtce:Comparison value="8" parameterRef="CCSDSPacketLength"/>
      </xtce:ComparisonList>
    </xtce:RestrictionCriteria>
  </xtce:BaseContainer>
</xtce:SequenceContainer>
```

Two Entries (if CCSDSSecH == 1) – plus header fields through inheritance (not shown)
Four Comparisons in RestrictionCriteriaplus the one in CCSDSTelemetryPacket (not shown)

Chapter 7 - Commands and Command Packets

- Command Pattern
  - Root CCSDSCommand
    - Has no arguments or any real content – an abstraction
  - Its CCSDSCommand/CommandContainer is called CCSDSCommandPacket
    - Extends CCSDSPacket and checks the cmd flag in the header
    - Slightly irregular use of XTCE (but not too bad)
  - Specific Mission Commands extend CCSDSCommand
    - Provide cmd arguments, command complete verifier
    - Its Mission Command Packets extends CCSDSCommand/CCSDSCommandPacket
      - Add entries
  - The mission command and packets are considered final and may not be extended

Chapter 7 – Command Example

```
<xtce:MetaCommand name="CAM0GetImagerInfo">
  <xtce:BaseMetaCommand metaCommandRef="CCSDSCommand"/>
  <xtce:ArgumentList>
    <xtce:Argument argumentTypeRef="CAM0ImagerNumberType" name="CAM0ImagerNumber"/>
  </xtce:ArgumentList>
  <xtce:CommandContainer name="CAM0GetImagerInfoPacket" shortDescription="Get information for a specific imager">
    <xtce:EntryList>
      <xtce:ArgumentRefEntry argumentRef="CAM0ImagerNumber"/>
    </xtce:EntryList>
    <xtce:BaseContainer containerRef="CCSDSCommandPacket">
      <xtce:RestrictionCriteria>
      <xtce:ComparisonList>
        <xtce:Comparison value="42" parameterRef="CCSDSSCID"/>
        <xtce:Comparison value="0" parameterRef="CCSDSVCID"/>
        <xtce:Comparison value="3" parameterRef="CCSDSAPID"/>
        <xtce:Comparison value="1" parameterRef="CCSDSPacketLength"/>
      </xtce:ComparisonList>
    </xtce:RestrictionCriteria>
    </xtce:BaseContainer>
  </xtce:CommandContainer>
</xtce:MetaCommand>
```

Command and its Packet
-CAM0GetImagerInfo
--CAM0GetImagerInfoPacket
One argument, & header entries (not shown)
-CAM0ImagerNumber
Four Comparisons (plus one in CCSDSCommandPacket not shown)

Chapter 7 – Draft Release

- Draft Release Contains
  - Presentations (explanatory)
  - Documentation – report describing GovSat rules
  - Rules – the rules table
  - Example – a full and realistic example

~Backups ~

Chapter 8 – A Quick Look at JAXB

- JAXB – Java Architecture for XML Binding
  - Will map your XML Schema (i.e. XTCE) to Java Classes
    - Has additional features including “binding” configuration file
- Used to implement most of GSFC XTCE prototype software
  - Plusses:
    - Java Generics and Collections well supported
    - Mapping tunable w/config file
    - Now part of official distribution
      - Everyone has it that downloads Java
    - Some support for DOM nodes
      - But not built on DOM (that I can tell!)
    - ...?
  - Misses:
    - Can’t seem to parse or build comments
    - Can’t convert any unmarshalled object to a DOM node
      - But seem to be able convert any DOM node to a marshalled object
    - ...?

Chapter 8 - JAXB SchemaTypes to Java Mapping

XML Schema DataType → Mapping to Java Type or Class (some)

Examples

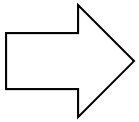
xsd:string → java.lang.String
xsd:integer → java.math.BigInteger
…
xsd:dateTime → javax.xml.dataType.XMLGregorianCaledar
…
xsd:double → double

Etc…

Chapter 8 - JAXB Ex. XTCE SchemaType to Java

XTCE SpaceSystemType

```
<complexType name="SpaceSystemType" mixed="false">
<complexContent mixed="false">
    <extension base="xtce:NameDescriptionType">
      <sequence>
        <element name="Header" type="xtce:HeaderType"
                 minOccurs="0"/>
        <element name="TelemetryMetaData"
                 type="xtce:TelemetryMetaDataType" minOccurs="0"/>
        <element name="CommandMetaData"
                 type="xtce:CommandMetaDataType" minOccurs="0"/>
        <element name="ServiceSet" minOccurs="0">
          <complexType>
            <sequence>
              <element name="Service" type="xtce:ServiceType"
                       maxOccurs="unbounded"/>
            </sequence>
          </complexType>
        </element>
        <element ref="xtce:SpaceSystem"
                 minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="operationalStatus"
                 type="token" use="optional"/>
    </extension>
  </complexContent>
</complexType>
```



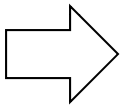
XTCE SpaceSystemType Java Class

```
public class SpaceSystemType extends NameDescriptionType
{
   protected HeaderType header;
   protected TelemetryMetaDataType telemetryMetaData;
   protected CommandMetaDataType commandMetaData;
   protected SpaceSystemType.ServiceSet serviceSet;
   protected List<SpaceSystemType> spaceSystem;
   protected String operationalStatus;
   public HeaderType getHeader() { return header;}
   public void setHeader(HeaderType value) { this.header = value;}
   public boolean isSetHeader() { return (this.header!= null);}

   public TelemetryMetaDataType getTelemetryMetaData() {
return telemetryMetaData;
   }
    public void setTelemetryMetaData(TelemetryMetaDataType value) {
      this.telemetryMetaData = value;
   }
   public boolean isSetTelemetryMetaData() {
      return (this.telemetryMetaData!= null);
   }
    public CommandMetaDataType getCommandMetaData() {
                  return commandMetaData;
    }
    public void setCommandMetaData(CommandMetaDataType value) {
      this.commandMetaData = value;
   }
   public boolean isSetCommandMetaData() {
      return (this.commandMetaData!= null);
   }

//  … DELETED FOR SPACE REASONS
}
```

Note: items edited to fit….

Chapter 8 – Simple JAXB XTCE Example


```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.Marshaller;
import org.omg.space.xtce.ObjectFactory;
import org.omg.space.xtce.SpaceSystemType;

public class SimpleSpaceSystem {

 public static void main(String[] args) {
 JAXBElement<SpaceSystemType> spaceRoot;
 JAXBContext jc;
 Marshaller marshaller;
 ObjectFactory factory;

 try {
  jc = JAXBContext.newInstance("org.omg.space.xtce");
  marshaller = jc.createMarshaller();
  marshaller.
    setProperty(
         "com.sun.xml.internal.bind.namespacePrefixMapper",
                              new XTCEPrefixMapper());
  marshaller.
     setProperty(Marshaller.JAXB_SCHEMA_LOCATION,
         "http://www.omg.org/space/xtce SpaceSystemV1.1.xsd");
  marshaller.
     setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
                    new Boolean(true));
  factory = new ObjectFactory();
  SpaceSystemType space = factory.createSpaceSystemType();
  space.setName("HelloWorld");
  spaceRoot = factory.createSpaceSystem(space);
  marshaller.marshal(spaceRoot, System.out);
 } catch (Exception e) {  e.printStackTrace(); }
}
```



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xtce:SpaceSystem xmlns:xtce="http://www.omg.org/space/xtce"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   name="HelloWorld"
   xsi:schemaLocation=
          "http://www.omg.org/space/xtce SpaceSystemV1.1.xsd"/>
```

Note: XTCEPrefixMapper not shown

Chapter 7 – Harder XTCE Items to Implement

- General NameReferences and NameReference Instances
  - All “refs” should point to something valid: you must implement this
  - Absolute addresses: /a/b/c and various forms of relative addresses: ../b/c, a/../b, etc...
  - “Path-less” nameRefs which may involve a partial tree search: c
  - Instances: Adds array syntax: /c[1], And aggregate syntax: /a/b/c.subField
    - And relates to some sort of “instance table”
- Container Inheritance, EntryList, IncludeConditions, etc...
  - Various general aspects of inheritance, EntryList addressing
    - Multiple levels, containerRefs that themselves have inheritance, and so on
- XTCE Type Inheritance
- Dynamic Container Match & NextContainer
  - Never been implemented yet that we are aware of...

(a few more not mentioned)

But wait, I have to implement all of those? Depends…

Chapter 7 – Native Implementation

- An implementation that uses XTCE internal to a toolchain
  - Gets packet stream on input, uses XTCE to pull-apart into host-side telemetry items
  - Or build cmds for uplink using definitions to supply args, values & format
- High degree of automation or self-configuration possible
  - Possibly implement every feature of XTCE
    - Thus potentially ingesting ANY XTCE file that is correct to self-configure software for any incoming data stream
  - But of course one can always constrain or restrict XTCE support if necessary
- Implementations?
  - Some aspects of GSFC prototype software falls into this category...
  - But never used to actually decommutate binary packet data (yet)
  - University of Bremen (Germany) implementation used XTCE to configure itself for decommutating binary packet file
    - Demo'd ... but source?

Chapter 7 – Exchange Implementations

- 100% XTCE Exchange between teams using same features
  - Agreement ahead of time on which features of XTCE to use
  - Depends somewhat on hardware of team members
    - If controlled or standardized between members: 100% exchange possible
- As these agreement don't exist, 100% exchange is not possible without COMMON AGREEMENTS
  - Ex. Extreme case: I support only telemetry, you support only Commands. Our XTCE files are 100% incompatible!
  - But even differing formats may offer partial exchange
    - NASA CCSDS & ESA PUS Mission were able to exchange the majority of telemetry metadata even though basic packet formats ultimately differ... but it took some work
- Implementations?
  - GSFC JWST Translator
  - CNES "Best" Suite
  - ESA Cryosat...
  - Several others... mostly on the "Customer" side – (NASAs of the world)

Chapter 7 – Ingredients in Successful Exchange

- Now:
  - Determine exchangees (team members), agree on common “flavor” of XTCE features
- Future:
  - Industry agreed upon flavors, pick the category you fall into...
- Develop common XTCE usage “constraints” to format
  - CCSDS is a “format” but stops at the header
  - You have to get down deeper Examples:
    - Polynomials, # of coefficients
    - Alarm levels
    - Floating point sizes and formats
    - Bit/byte ordering...
    - Etc..., etc..., etc...
- Implement to these constraints & 100% exchange is possible between team members, institutions, and so forth

Chapter 7 – Automating Exchange?

- Write XTCE constraints in a computer processable manner
- Process constraints to configure validation software
  - Use in conjunction with XML Parsing to validate XML in “your” XTCE
- Process constraints to configure “XTCE builder” software
  - An API that only allows XTCE files to be constructed in “your” flavor
- Are all constraints describable in a computer processable manner?
  ???
- Is this an additional area of standardization?